QUEUES AND PRIORITY QUEUES

Problem Solving with Computers-II

How is PA03 going?

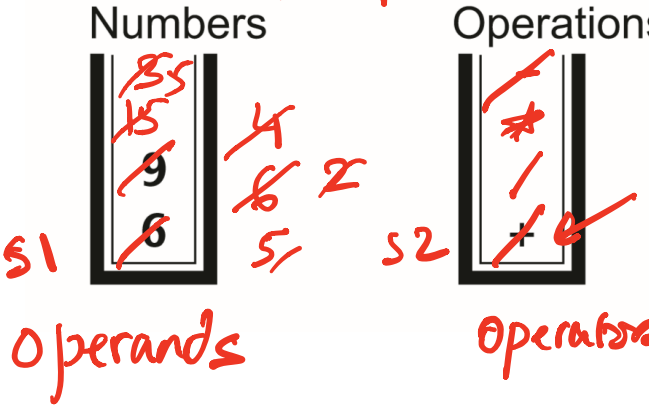
A. Done
B. On track to finish
C. Having trouble with the checkpoint (design)
D. Just started
E. Haven’t started

Evaluating a fully parenthesized infix expression

evaluate the sub expression in the 2 stacks

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



9 + 6

15 / 3

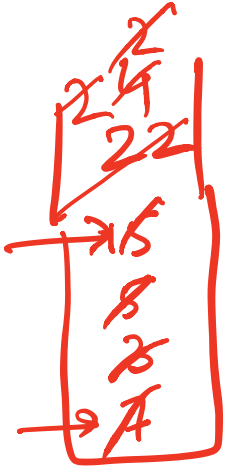
5 * 2 = 10

Lab 05, part2 :
Evaluating post fix expressions using a single stack

Postfix: 7 3 5 * + 4 2 / -

Infix: (7 + (3 * 5)) – (4 / 2)

20



One stack

5 * 3

15 + 7

20

The Queue Operations

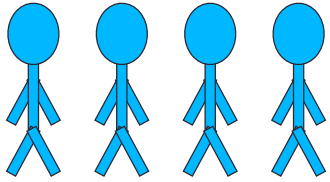
- A queue is like a line of people waiting for a bank teller.
- The queue has a **front** and a **rear**.

STL queue

① push()
// insert()
// enqueue()

③ front()

④ empty()



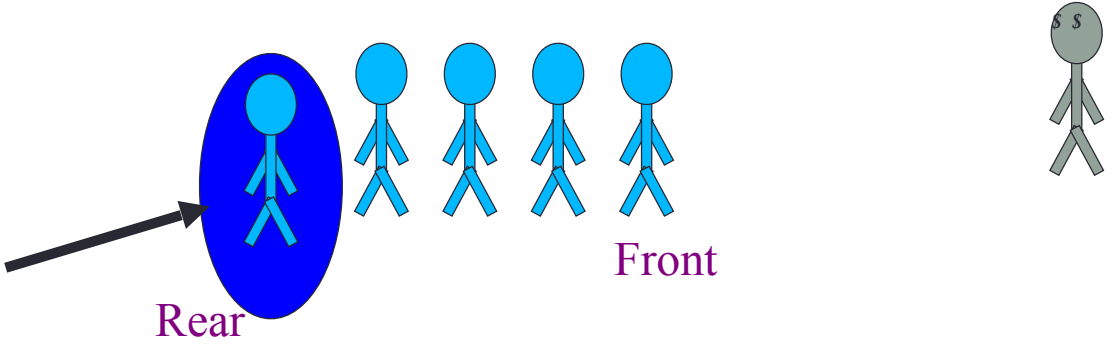
Rear

Front

② pop()
// delete front
// dequeue

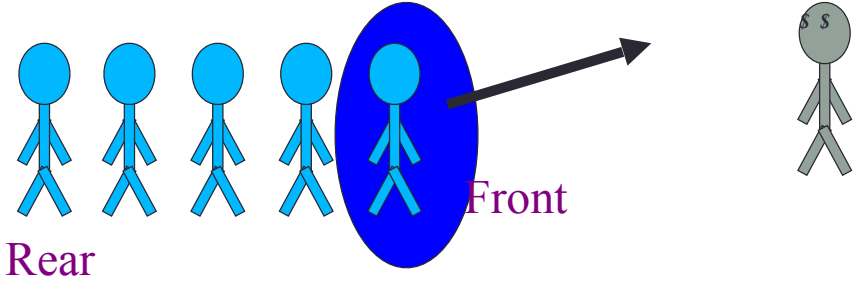
The Queue Operations

- New people must enter the queue at the rear. The C++ queue class calls this a **push**, although it is usually called an **enqueue** operation.



The Queue Operations

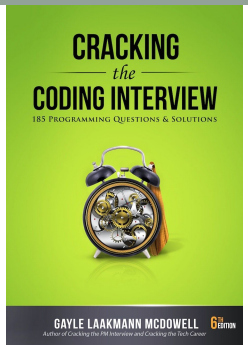
- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a **pop**, although it is usually called a **dequeue** operation.



The Queue Class

- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>
class queue<Item>
{
public:
        queue( );
        void push(const Item& entry);
        void pop( );
        bool empty( ) const;
        Item front( ) const;
        …
```



Queue via stacks

Implement a MyQueue class which implements a queue using ~~two~~ stacks

```
class MyQueue {
    public:
        void push(int item);
        void pop();
        int front();
        bool empty();
    private:
        stack<int> s1;
        stack<int> s2;
};
```

Priority Queues or Heaps

	Min-Heaps	Max-Heap	BST (balanced)
Insert :	O(log N)	O(log N)	O(log N)
Min:	O(1)	—	O(log N)
Delete Min:	O(log N)	—	O(log N)
Max	—	O(1)	O(log N)
Delete Max	—	O(log N)	O(log N)

Choose heap if you are doing repeated insert/delete/(min OR max) operations

Applications:

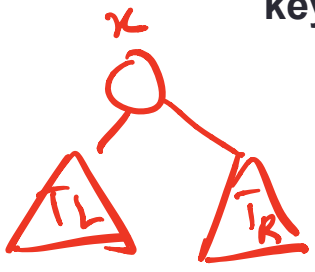
- Efficient sort
- Finding the median of a sequence of numbers
- Compression codes Huffman code

Heaps as binary trees

Binary Trees
BST Heap

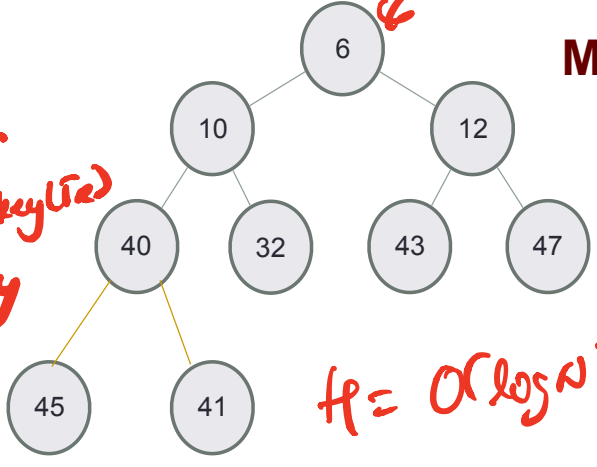
- **Rooted binary tree that is as complete as possible**
- **In a min-Heap, each node satisfies the following heap property:**

  key(x)<= key(children of x)



keys(T_L) ≤ x ≤ key(T_R)
BST property

min is the root



Min Heap with 9 nodes

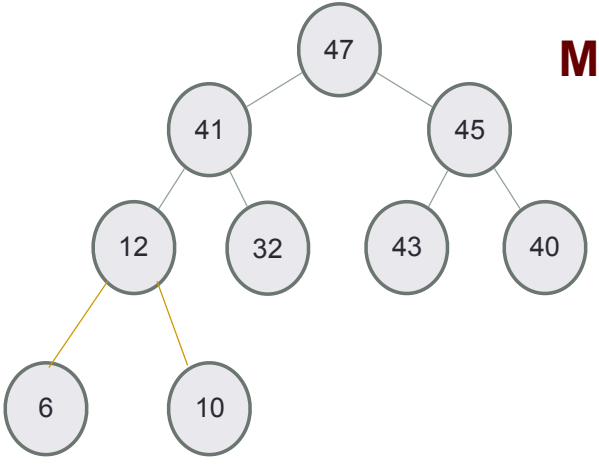
$H = O(\log N)$

Where is the minimum element?

Heaps as binary trees

- **Rooted binary tree that is as complete as possible**
- **In a max-Heap, each node satisfies the following heap property:**

  key(x)>= key(children of x)



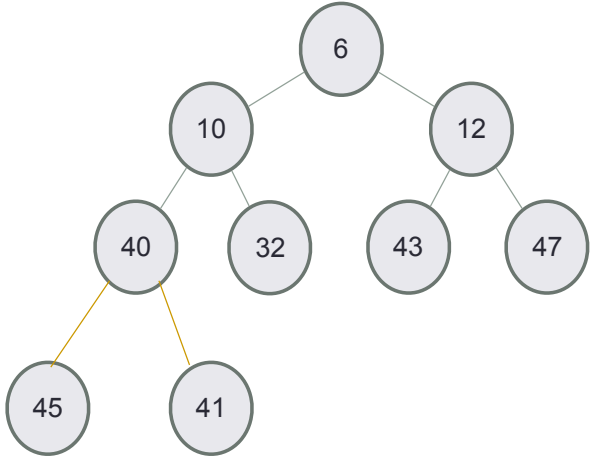
Max Heap with 9 nodes

Where is the maximum element?

Identifying heaps

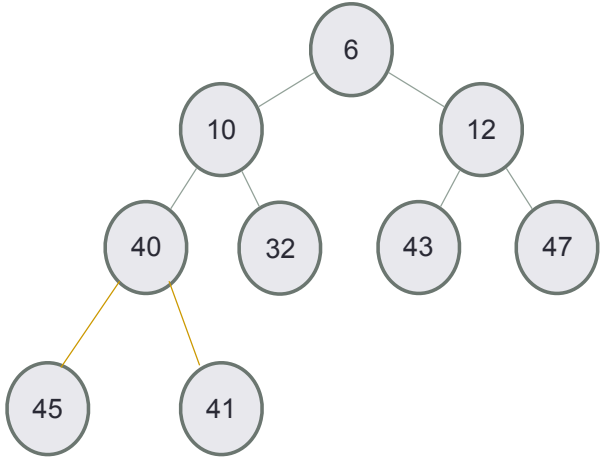
Starting with the following min Heap which of the following operations will result in something that is NOT a min Heap

A. Swap the nodes 40 and 32
B. Swap the nodes 32 and 43
C. Swap the nodes 43 and 40
D. Insert 50 as the left child of 45
E. C&D



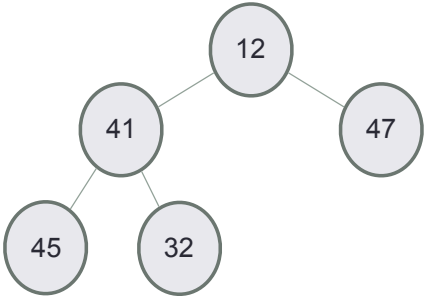
Structure: Complete binary tree

A heap is a complete binary tree: Each level is as full as possible. Nodes on the bottom level are as far left as possible

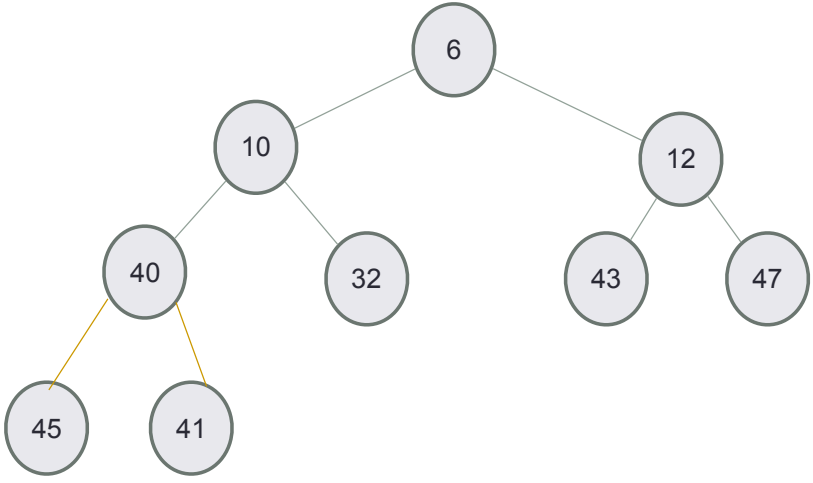


Insert 32 into a heap

- **Insert key(x) in the first open slot at the last level of tree (going from left to right)**
- **If the heap property is not violated - Done**
- **Else: while(key(parent(x))>key(x)) swap the key(x) with key(parent(x))**

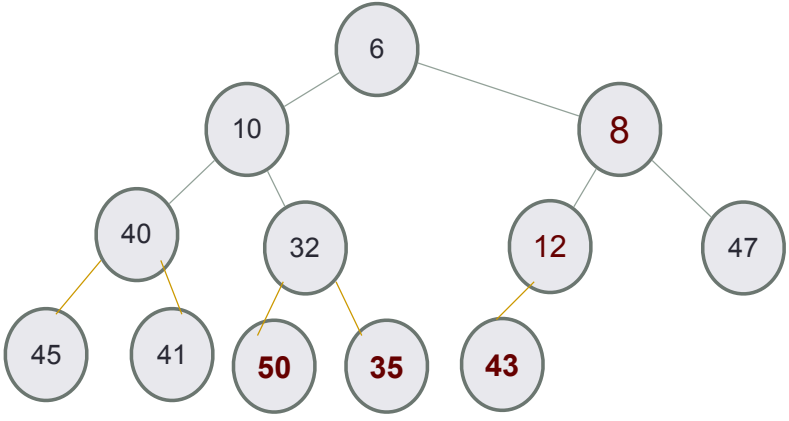


Insert 50, then 35, then 8



Delete min

- **Replace the root with the rightmost node at the last level**
- **“Bubble down”- swap node with one of the children until the heap property is restored**



Next lecture

- **Under the hood of priority queues**